Downloading Programs

On a first-name basis

Aug 28, 1990

Previous application management

Application page programs heretofore have been allocated manually in the VME local stations. One found some free space in non-volatile memory and downloaded the code for an application page into it. Then the starting address was entered on the index page to associate a page with the program.

Difficulties with that scheme arose when the number of downloaded programs got to be large. (Once upon a time, there were but four.) It was hard to manage the memory available efficiently. One tended to select starting addresses on 4K-byte boundaries just to keep it simpler, resulting in fragmentation.

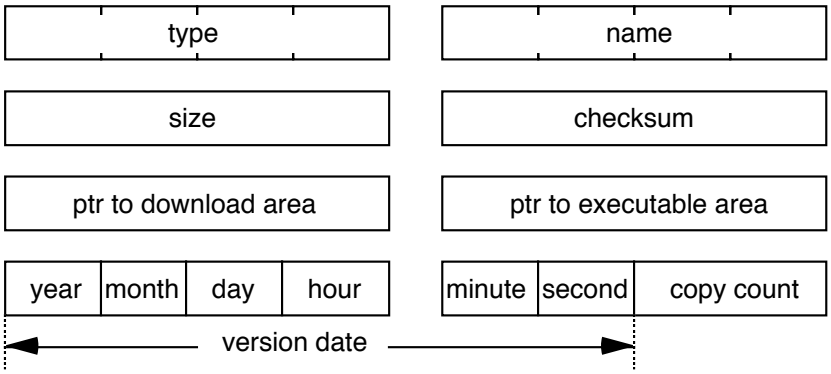
Another difficulty arose when downloading to another system. One must take care that a previous version of the program was not currently executing on that system. If it were, a crash would likely result as the new code comes raining down upon the executing code. This crash is not easily recoverable, since the system will try to recall the same page after it aborts. After 15 times, the system aborts more thoroughly by exiting to SBug, requiring local manual intervention at that station to recover.

A third problem with the previous scheme was that updating new versions of code required copying to the various local stations one at a time. Only strict adherence to conventions can insure that the program is downloaded in the same area of memory on separate nodes. (There is no system requirement for such.) Different nodes may have different needs for a repertoire of page applications.

A fourth problem was lack of checking to insure that a program residing in non-volatile memory was not corrupted by an errant happenstance in a local station.

Named-based programs

By supporting communications by program name, we insulate the user from having to deal with manual memory allocation and at the same time, we can be free of concern for locating a given program at the same address in multiple nodes. A new system table called `CODES` is used to hold information about each downloaded program. Its layout is as follows:

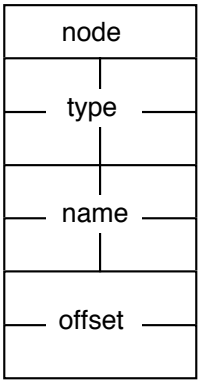


The 4-character program `type` distinguishes programs used for different purposes. Values such as `PAGE`, `LOOP`, `TASK` can be used to distinguish page application programs, local application programs and task initialization programs. The 4-character program `name` serves to distinguish programs of the same `type`. The program `size` is next, followed by the

longword `checksum`, which is a sum of words. The `download ptr` is a ptr to the allocated area in non-volatile memory. This is followed by the `execution ptr`, the ptr to the area in on-board ram where the program was copied for execution. The `date` serves to identify the version of the program. The `copy count` shows the number of times the program was copied into on-board ram for execution. It is cleared when a new version is downloaded.

New listype to support downloading

Support for name-based downloading requires a new listype (#76). The ident format is as follows:



Both the `type` and `name` codes are included to fully identify the program being referenced. The `offset` is a longword to permit downloading of large programs of the future, when 16-megabyte ram chips are commonplace. Special `offset` values are used for beginning and ending the download process.

To download a program, an `offset` value of –1 is used to send the `type`, `name` and `size` information to the receiving node. If the receiving node has the program in its `CODES` table, it deletes it and frees its memory, establishing a new incomplete entry that includes a –1 checksum value. (If the program was not currently registered, it must find a vacant table entry and initialize it.) At this point, since the `checksum` is –1, the program is no longer eligible to be copied for execution successfully, although a currently-executing version can still be running, since it runs in on-board memory, which is separate from the download area. The `size` is used to allocate memory to receive the downloaded program, and that `download ptr` is placed in the entry.

Settings are delivered with `offset` values ≥ 0 (but not +1) to fill the download area with the program. After that is finished, which may require multiple settings, a special setting is made with the `offset` value of +1. The data sent with that setting is the `checksum` longword and optionally the version `date`. The next time the page is called up, in the case of a `PAGE`-type program, the new program will be checked for corruption using the `checksum` longword and copied into on-board ram for execution. In the case of a `LOOP`-type program, the next time the closed loop is enabled, the new version will be used. If such a program is enabled during the download operation (and executing out of on-board ram), it will automatically be disabled and automatically re-enabled on the next cycle, resulting in automatic and orderly transition to the new version.

A data request using this listype deals with the same data as the setting does. Notice that no program header is required, as all the `size` and `name` information is carried in the `CODES` table entry. The data returned for an `offset` value = –1 is the contents of the `CODES` table entry itself. An `offset` value ≥ 0 returns arbitrary portions of the program code.

***CODES** Table Management*

An enhanced version of the download application page allows inspection of the various CODES table entries. One can disable a downloaded program from further invocations and free its memory, much as one would delete a file in a disk operating system to get more room for storing new files.

To maintain the free space in one contiguous piece, the system moves allocated blocks around at system reset time to eliminate fragmentation before any programs are put to use. It has no side effects, since only the CODES table knows where they are. Any other references to these programs are by name.